



## 02. Elementos básicos de C++

Programación - 1º DAM

Luis del Moral Martínez

versión 20.10

Bajo licencia CC BY-NC-SA 4.0



# Contenidos del tema

1. Entornos de desarrollo
2. Primer programa en C++
3. Proceso de ejecución de un programa
4. Estructura y elementos de un programa
5. Tokens, identificadores y palabras reservadas
6. Tipos de datos en C++
7. Constantes, variables y asignaciones
8. Entrada y salida por consola
9. Espacios de nombres
10. Depuración de un programa

# 1. Entornos de desarrollo

## ¿Qué es un entorno de desarrollo?

- Un **entorno de desarrollo**, o IDE, permite un desarrollo de software avanzado
- Posee **herramientas** que permiten compilar, ejecutar y depurar programas
- Permiten **resaltar** la sintaxis y nos ayudan a **detectar errores** en el código
- Existe una gran diversidad de entornos: Eclipse, NetBeans, Visual Studio, Code::Blocks...
- La mayoría son multiplataforma (Linux, Windows y MacOS)
- Otra opción sería instalar el compilador y trabajar con un editor, pero es menos versátil

# 1. Entornos de desarrollo

## Instalación del entorno de desarrollo Code::Blocks

- **Paso 1:** Acceder a la web de [Code::Blocks](#)
- **Paso 2:** Descargar el fichero que contiene también el compilador *Mingw* (G++ y GDB)
  - [codeblocks-17.12mingw-setup.exe](#)
  - En el caso de Linux se puede descargar la opción para tu distro y después instalar g++
- **Paso 3:** Instalar la aplicación, dejando todas las opciones por defecto

# 1. Entornos de desarrollo

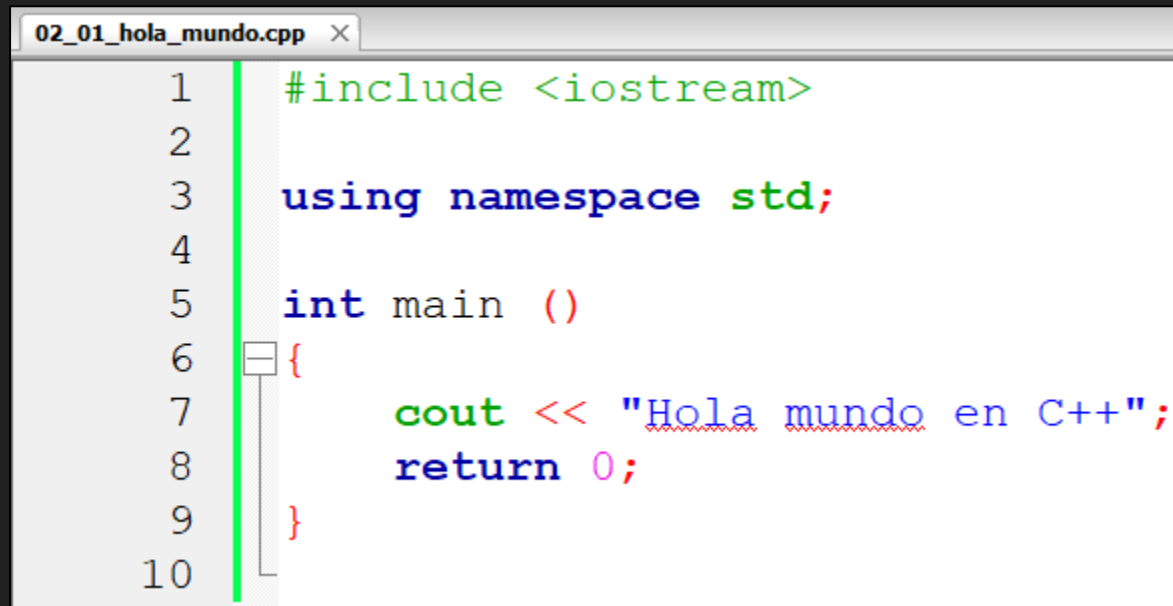
## Instalación del compilador g++ en Linux

- **Paso 1:** Instalar el compilador g++
  - `sudo apt-get install g++`
- **Paso 2:** Instalar y configurar el editor que queramos utilizar para editar el código
  - Visual Studio Code, Atom, Geany, Gedit...

## 2. Primer programa en C++

### Creando el primer programa en C++

- Los ficheros de código de un programa en C++ tienen la extensión .cpp
- En **Code::Blocks**, o el editor que vayas a utilizar, abre el fichero **02\_01\_hola\_mundo.cpp**

A screenshot of a code editor window titled '02\_01\_hola\_mundo.cpp'. The editor displays a C++ program with line numbers 1 through 10 on the left. The code is as follows:

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main ()
6  {
7      cout << "Hola mundo en C++";
8      return 0;
9  }
10
```

# 3. Proceso de ejecución de un programa

## ¿Cómo ejecutamos el programa?

- En Code::Blocks

- Pulsar el botón de ejecución (ubicado en la barra de herramientas):



- Por **consola**:

- Compilar el programa: `g++ -o programa.cpp`
  - Ejecutar el programa `./programa.exe` (o `programa.exe` si estamos en el cmd de Windows)

- En ambos casos, se mostrará el resultado del programa por consola:

```
Hola mundo en C++  
Process returned 0 (0x0)   execution time : 0.050 s  
Press any key to continue.
```

# 4. Estructura y elementos de un programa

## Estructura de un programa en C++

- Un programa se compone de:
  - La función **main**, que es la función principal del programa (obligatoria)
  - Funciones adicionales (definidas por el usuario o de la librería estándar de C++)
  - Directivas de preprocesador (**#include**, **using**...)
  - Declaraciones globales
  - Las funciones contienen a su vez **comentarios** y **sentencias**



# 4. Estructura y elementos de un programa

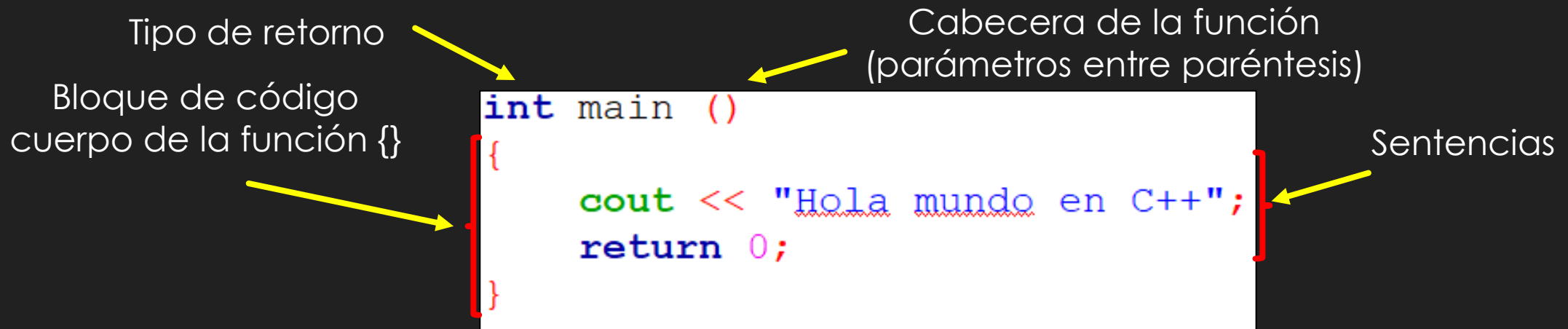
## Elementos de un programa en C++: directivas del preprocesador

- Son comandos para configurar el preprocesador
- El preprocesador se encarga de procesar el código fuente antes de compilarlo
- Permite incluir en el código otros archivos (cabecera)
  - **#include**: permite incluir otro archivo en el programa (importar librerías y funciones)
  - **Archivo de cabecera (iostream)**: proporcionan funciones o herramientas a nuestro programa
    - **#include <iostream>**: incluye la biblioteca de funciones de entrada/salida para usarla en el programa (cout...)
  - **using**: indica que vamos a utilizar un espacio de nombres
    - **using namespace std**: indica que las sentencias del programa se incluyen en el espacio de nombres *std*

# 4. Estructura y elementos de un programa

## Elementos de un programa en C++: la función main()

- Es la función principal del programa
- Constituye el **punto de entrada** al programa
- Cualquier otra función se invoca desde main()



# 4. Estructura y elementos de un programa

## Elementos de un programa en C++: los comentarios

- Los comentarios permiten explicar una sentencia o un bloque de código
- Existen dos estilos:
  - **Estilo C:** `/* Esto es un comentario */` (permite comentarios de bloque de código)
  - **Estilo C++:** `// Esto es un comentario`

# 5. Tokens, identificadores y palabras reservadas

## Tokens

### ■ **Identificadores**

- Permiten poner nombre a las variables: nombre\_clase, elemento, numero...
- Secuencia de letras, dígitos y subrayados (\_), pero el primer carácter siempre es una letra
- C++ es sensible a las mayúsculas (case-sensitive)

### ■ **Palabras reservadas** (keywords)

- Son palabras reservadas del lenguaje: void, main, int, char, return... [\[más información\]](#)

### ■ **Signos de puntuación y separadores**

- ! % ^ & \* ( ) - + = { } ~ [ ] / : ' < > ? , . / "

# 6. Tipos de datos en C++

## Resumen de los principales tipos de datos

Tipo	Ejemplo	Tamaño (bytes)	Rango (min – max)
char	'C'	1	0 – 255
short	-15	2	-128 – 127
int	1024	2	-32768 – 32767
unsigned int	42325	2	0 – 65535
long	262144	4	-214783648 – 2147483637
float	10.5	4	$3.4 \times 10^{-38}$ – $3.4 \times 10^{38}$
double	0.00045	8	$1.7 \times 10^{-308}$ – $1.7 \times 10^{308}$
long double	1e-8	8	Igual que double
bool	true	1	true o false

# 6. Tipos de datos en C++

## Otros tipos de datos

- **void**
  - Indica que no se toma ningún valor (nada).
  - Suele utilizarse para declarar una función que no va a tener parámetros
  - También indica que la función no devuelve ningún valor
- **Nota sobre los caracteres (tipo char)**
  - Se utiliza el conjunto de caracteres ASCII (-128...127) para representar un carácter
  - Se escriben entre comillas simples ''

# 7. Constantes, variables y asignaciones

## Constantes

- Se declaran con la palabra reservada **const** (**const int constante = 3;**)
- Se asigna un valor en el momento de la declaración y este no se puede modificar
- Las constantes no cambian durante la ejecución del programa
- Tipos de constantes:
  - **Literales**: 123456, 3.14, 'A'
  - **Simbólicas**: #define PI 3.141592
  - **Enumeraciones**: enum Colores {Rojo, Naranja, Amarillo, ...} (Rojo-0, Naranja-1, Amarillo-2...)
- **const** ocupa espacio de ejecución, mientras que **#define** es reemplazado al compilar

# 7. Constantes, variables y asignaciones

## Variables

- Es una posición en memoria en la que se almacena un valor de un tipo de dato determinado
- Este valor puede ser modificado (en cambio, una constante no puede ser modificada)
- Las variables pueden almacenar cualquier tipo de dato: cadenas, números, estructuras...
- El nombre de las variables tiene que ser un **identificador** válido
- Las variables se pueden declarar en cualquier punto del programa



# 7. Constantes, variables y asignaciones

## Asignación e inicialización de variables

- Al crear una variable, esta no tiene ningún valor asociado
- Si se utiliza sin un valor inicial, el programa podría funcionar incorrectamente
- Las variables pueden inicializarse en el momento de su declaración o antes de usarlas
- **Definición de una variable:** tipo identificador; (`int numero;`)
- **Definición e inicialización de una variable:** tipo identificador = valor; (`int numero = 5;`)
- **Agrupación de definiciones:** `int variable1, variable2, variable3;`
- **Agrupación con inicialización:** `int variable1 = 5, variable2 = 3, variable3 = 7;`

# 7. Constantes, variables y asignaciones

## Ámbito de las variables

- Dependiendo de dónde se defina una variable, se podrá usar en todo el programa o no
- Una variable puede existir dentro de un bloque de código (definido entre llaves { y })
- **Variable local:**
  - Existe dentro de una función y solo son visibles en esa función
  - Dos o más funciones pueden tener una variable con el mismo nombre (no se ven entre sí)
  - Las variables no existen en memoria hasta que no se ejecuta la función
- **Variable global:** global a todo el código (suele ser una mala práctica de programación)

# 7. Constantes, variables y asignaciones

## Un poco de práctica

- Abre ahora los siguientes programas y ejecútalos:
  - **02\_02\_constantes.cpp**
  - **02\_03\_variables.cpp**
  - **02\_04\_ambito.cpp**
- Observa su funcionamiento y comenta los resultados con tus compañeros

# 8. Entrada y salida por consola

## Usando la entrada y salida

- En todos los ejemplos anteriores hemos hecho uso de la biblioteca `iostream` (entrada y salida)
- Esta biblioteca permite **imprimir** un valor por consola o **pedírselo** al usuario
- No olvidar **`using namespace std`** (o de lo contrario se usarán como **`std::cin`** y **`std::cout`**).
- **Entrada (`cin`):**
  - Permite leer el valor de una variable por consola (stdin): **`cin >> variable;`**
- **Salida (`cout`):**
  - Permite imprimir un valor de una variable o un mensaje por consola (stdout): **`cout << "Hola mundo"`**

# 8. Entrada y salida por consola

## Usando la entrada y salida

- Con cout puedes imprimir una cadena de caracteres con comillas dobles `cout << "hola";`
- Estas cadenas de caracteres pueden incluir una secuencia de escape: `\n`, `\t...`
- Se puede imprimir una nueva línea con `endl`:
  - `cout << "hola" << endl;`
- Se puede mezclar una cadena de caracteres con una variable y endl (usando más <<):
  - `cout << "Valor de la variable: " << variable << endl;`
- Abre el fichero `02_05_entrada_salida.cpp`

# 8. Entrada y salida por consola

## Secuencias de escape más habituales

Secuencia de escape	Significado
<code>\n</code>	Nueva línea
<code>\r</code>	Retorno de carro
<code>\t</code>	Tabulación
<code>\v</code>	Tabulación vertical
<code>\\</code>	Barra inclinada \
<code>\'</code>	Comilla simple
<code>\"</code>	Doble comilla
<code>\?</code>	Signo de interrogación

# 9. Espacios de nombres

## ¿Qué es un espacio de nombres?

- Permite agrupar código bajo un nombre (aunque es opcional)
- El nombre tiene que ser un identificador básico: `espacio_de_nombres`
- Uso:
  - `namespace::funcion();`
  - `namespace::constante`
- Aquí también podemos utilizar `using namespace espacio_de_nombres;`
- Abre el fichero `02_06_namespace.cpp`

# 10. Depuración de un programa

## Concepto de depuración

- Un programa rara vez funciona a la primera
- Los errores deben ser **detectados**, **aislados** y **corregidos**
- El proceso de corrección de un programa se denomina **depuración**
- El depurador permite analizar el programa para **detectar los errores**
- La depuración no es sinónimo de **probar el software**
- Un buen desarrollo de software incluye también una **batería de pruebas del código (testing)**



# 10. Depuración de un programa

## Tipos de errores

- **Errores de sintaxis:** errores de incumplimiento de la sintaxis del lenguaje
- **Errores lógicos:** errores en el diseño del algoritmo
- **Errores de regresión:** se crean accidentalmente al corregir errores lógicos
- **Mensajes de error:** los muestran los compiladores en la fase de compilación
  - **Errores de sintaxis, warnings** (condiciones sospechosas en el código) o **errores fatales** (poco comunes)
- **Errores en tiempo de ejecución:** se detectan durante la ejecución del programa
- **Pruebas:** las pruebas (o tests) permiten verificar el correcto funcionamiento del programa

## 10. Depuración de un programa

## Depurando un programa en Code::Blocks (GDB)

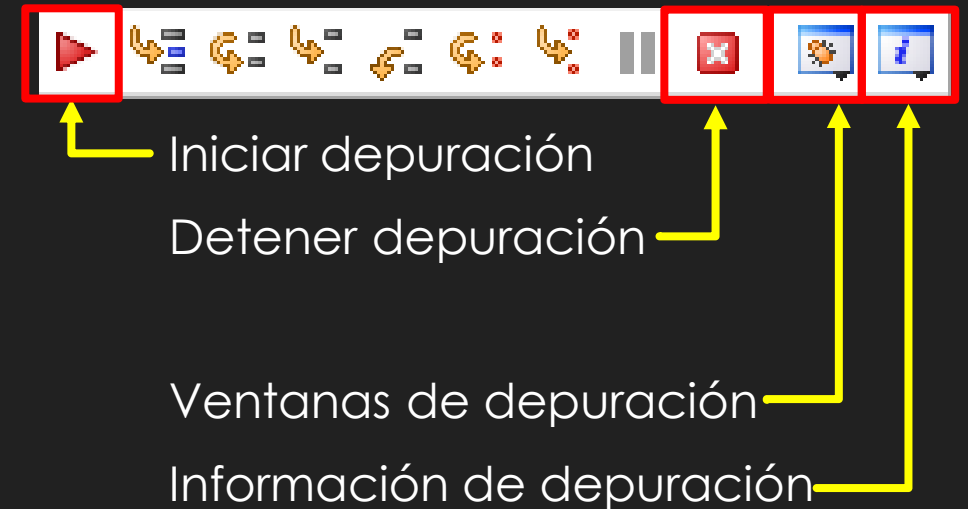
- Code::Blocks no permite depurar un fichero de código independiente
- Para ello tendremos que crear un **proyecto**:
  - **Paso 1:** File > New > Project...
  - **Paso 2:** En **Build Targets** seleccionamos **Console** y elegimos el lenguaje C++
  - **Paso 3:** Ponemos un **nombre** al proyecto y lo guardamos en una ruta
- Podemos depurar el programa usando las herramientas de depuración
- Prueba a depurar alguno de los ejemplos anteriores



# 10. Depuración de un programa

## Conceptos de depuración

- **Breakpoint:** punto de ruptura (detener depuración)
- **Continue:** continuar hasta el siguiente **breakpoint**
- **Depuración paso a paso:**
  - Permite ejecutar el código **instrucción a instrucción**
- **Ventanas de depuración:**
  - breakpoints, registros, pila, memoria, variables...
- **Información de depuración:**
  - Librerías, estado de la pila...



# Créditos de las imágenes y figuras

## Cliparts e iconos

- **Obtenidos mediante la herramienta web [IconFinder](#)** (según sus disposiciones):
  - Diapositiva 1
  - Según la plataforma IconFinder, dicho material puede usarse libremente (free comercial use)
  - A fecha de edición de este material, todos los cliparts son free for comercial use (sin restricciones)

## Resto de diagramas y gráficas

- Se han desarrollado en PowerPoint y se han incrustado en esta presentación
- Todos estos materiales se han desarrollado por el autor
  - Si se ha empleado algún icono externo, este se rige según lo expresado anteriormente